The Sparse Tableau Approach to Network Analysis and Design

GARY D. HACHTEL, MEMBER, IEEE, ROBERT K. BRAYTON, AND FRED G. GUSTAVSON

Abstract—The tableau approach to automated network design optimization via implicit, variable order, variable time-step integration, and adjoint sensitivity computation is described. In this approach, the only matrix operation required is that of repeatedly solving linear algebraic equations of fixed sparsity structure. Required partial derivatives and numerical integration is done at the branch level leading to a simple input language, complete generality and maximum sparsity of the characteristic coefficient matrix. The bulk of computation and program complexity is thus located in the sparse matrix routines; described herein are the routines OPTORD and 1-2-3 GNSO. These routines account for variability type of the matrix elements in producing a machine code for solution of $Ax=b$ in nested iterations for which a weighted sum of total operations count and round-off error incurred in the optimization is minimized.

I. Introduction

COMPUTER-AIDED design of electrical networks has recently been undergoing a rapid evolution due to the development of highly efficient methods for solving systems of linear equations where the matrix of coefficients is sparse [1]–[4]. These new techniques, when fully absorbed in a computer-aided design program, drastically affect even the initial formulation of the problem.

In this paper we describe a research investigation aimed at designing and implementing a computer network design program (NDP) for automated network optimization which fully incorporates the sparse matrix methods. The formulation is unorthodox yet simple and is applicable to any system described mathematically by a set of algebraic and differential equations. A general nonlinear electrical network is such a system, and it follows that the basic procedure for solving for the currents and voltages in a network can be viewed in terms of a simple mathematical procedure: Gaussian elimination. This has both a simplifying and unifying effect and makes possible a simple yet truly general-purpose computer program. By combining the concept of variability type with sparse matrix techniques, this program can achieve practically optimum efficiency by choosing the order of elimination so as to minimize the total operations count required for simulation and/or automated optimization.

In this section we give an overview of our method for orientation purposes. Basic to our approach is the concept of a tableau which includes all network information in a nonreduced form.

Manuscript received May 27, 1970; revised August 10, 1970.

The authors are with the IBM T. J. Watson Research Center, Yorktown Heights, N. Y. 10598.

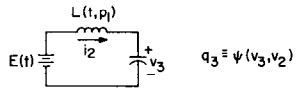


Fig. 1. Nonlinear network.

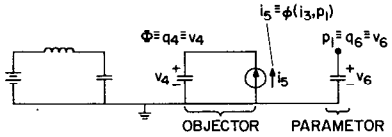


Fig. 2. Augmented network.

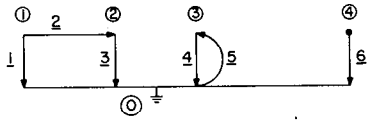


Fig. 3. Node-branch labeling for the directed graph.

For illustration consider the automatic design of the network of Fig. 1. The nonlinear elements were chosen to demonstrate that there are no restrictions on element type. In general, we consider design objectives of the class

$$\underset{p_m \le p \le p_M}{\text{minimize}}\ \Phi \equiv \int_0^T \phi(w, p, t)\, dt \tag{1}$$

where Φ represents a generalized scalar objective functional and ϕ an objective function of the unknown vector w and the vector of designable parameters p. In Fig. 1, $\phi = \phi(i_3, p_1)$. It is convenient in automated network design to add these quantities directly to the tableau of unknowns. This is done (Fig. 2) by augmenting the original network with a unit capacitance driven by a dependent current source of value ϕ (the integrand in (1)) and another unit capacitance connected only to the reference node. The reason for this last augmentation has to do with adjoint sensitivity computation [5] and will be made clear in Section II. The first augmentation (cf., Fig. 2) is called herein an *objector* and the second a *parametor*. The directed graph of the augmented network is shown in Fig. 3 with integer node and branch labels. Fig. 4 shows the tableau of algebraic and differential equations which describe the network, where V, i, v stand

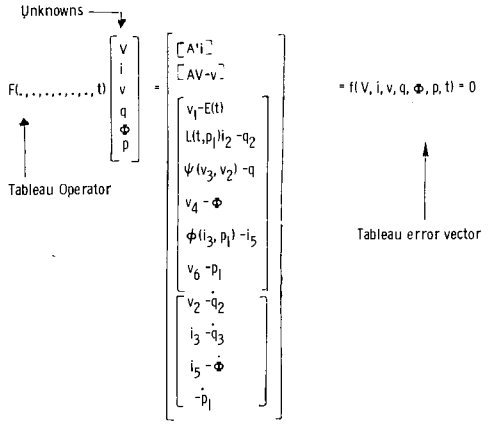


Fig. 4. Tableau operator, unknown vector, and tableau vector.

for the vectors[1] of node voltages, branch currents, and branch voltages, respectively. Energy storage is denoted by the generic term q which stands, in network applications, either for magnetic flux or electronic charge. However, since the objector and parametor charges are of special interest, the labels Φ and p are retained for these elements.

The tableau operator $F(.,.,.,.,.,.,.,.,t)$ (Fig. 4) is an algebraic–differential operator which operates to the right on the unknown vector $w \equiv \text{col}\,(V, i, v, q, \phi, p)$ to form the tableau error vector $f(w, t)$. The network equations may thus be stated in the form

$$f(w(t), t) \equiv f(V, i, v, q, \Phi, p, t) = 0, \qquad t \in [0, T]. \quad (2)$$

The tableau matrix $F_l(w, t)$ is the linearization of the tableau operator F around the operating point w, i.e.,

$$F_l(w, t) \equiv \frac{\partial f(w, t)}{\partial w}. \quad (3)$$

The tableau matrix for the network of Fig. 1 is illustrated in Fig. 5. Figs. 4 and 5 show that (for this case) the first ten rows represent the Kirchoff's current ($A'i=0$) and voltage ($AV-v=0$) laws. The matrix A is the branch versus node incidence matrix [6], which has the advantage of being directly obtainable from the network description input data cards (which also establish the labeling of Fig. 3.) Rows 11–16 of f (Fig. 4) state the branch constitutive relations (one per branch) of the network, while rows 17–20 give the dynamic relations describing the generalized energy storage elements (e.g., $i_3 - \dot{q}_3 = 0$ (row 18)).

For general networks, the tableau matrix F_l has the form of Fig. 6, wherein for network applications submatrices A, B, and E have all elements ± 1 or 0. Note F_l is very sparse, averaging in general about three nonzero elements per row.

Note that the optimization problem (1) for any system of ordinary differential equations (ODE) of the form $g(\dot{q}, q, p, t) = 0$ can be put in the form of Fig. 6 by the following

[1] A lower case letter is a column vector unless otherwise specified, "′" denotes the transpose operation. Thus v' is a row vector.

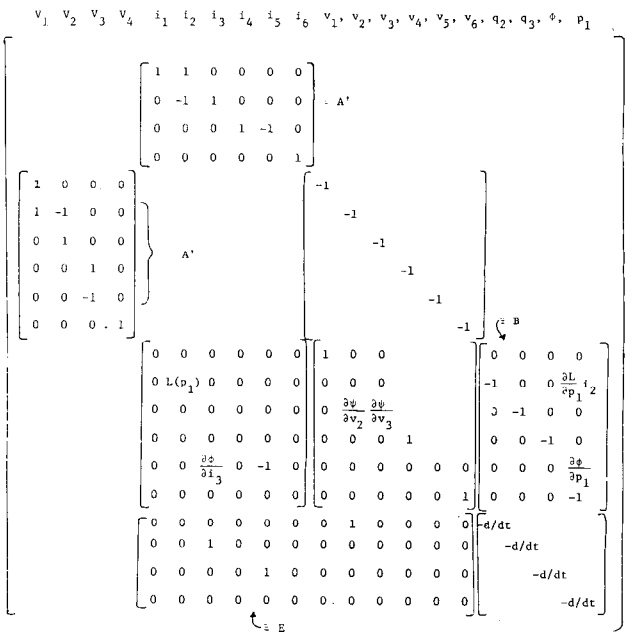


Fig. 5. Tableau matrix.

0	A′	0	0
A	0	-I	0
0	Z	Y	B
0	E		$-d/dt$

Fig. 6. Form of general tableau matrix.

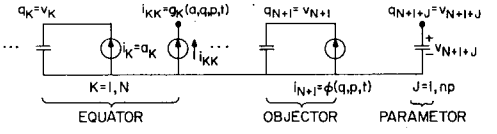


Fig. 7. Network analog of general ODE optimization problem.

substitution:

$$f(a, q, \Phi, p, t) \equiv \begin{bmatrix} g(a, q, p, t) \\ a \quad -\dot{q} \\ \phi \quad -\dot{\Phi} \\ \quad -\dot{p} \end{bmatrix} = 0, \qquad t \in [0, T]. \quad (4)$$

The network analog of the general ODE optimization problem is shown in Fig. 7, wherein each of the N components g_K, $K=1, N$, of the system $g(a, q, p, t)=0$ is represented by an *equator*, i.e., a unit capacitance driven by a current source, $i_K = a_K$, and a floating current source of value $i_{KK} = g_K(a, q, p, t)$. The objector and parametors (one for each of the np designable parameters) are as previously defined for the network problem.

The tableau approach to time-domain simulation is to discretize, at the branch level, the derivative operator d/dt. The nth discrete time point is denoted by t_n and $w_n \equiv w(t_n)$. Thus the last $nq+1+np$ rows (nq charges, 1 objector, np

parametors) have d/dt replaced by the kth-order backward differentiation formula [7]. For example, for charges

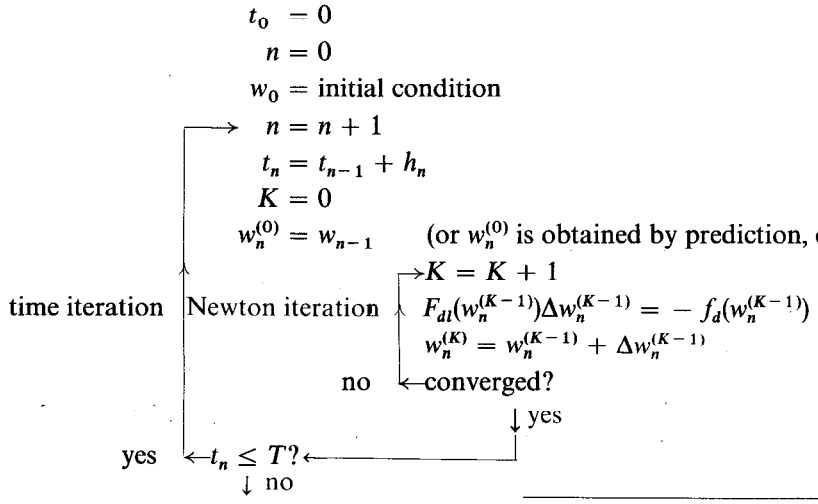
$$\dot{q}(t_n) \equiv \frac{dq_n}{dt} \equiv -\left(\sum_{j=0}^{k} \alpha_j q_{n-j}\right)/h \tag{5}$$

where $h \equiv t_n - t_{n-1}$. This transforms the tableau operator F, the tableau error vector f, and the tableau matrix F_l, into their discrete time forms F_d, f_d, and F_{dl}. Here f_d is obtained from f by replacing $\dot{q}$, $\dot{\Phi}$, and $\dot{p}$ by sums of the form (5), and F_{dl} is obtained from F_l (Figs. 5 and 6) by replacing d/dt by $-\alpha_0/h$.

Thus the time-domain simulation problem (2) is transformed into that of solving the difference equations

$$w_0 \equiv w(t_0) \equiv w(0)$$
$$F_d(\cdot, p, t_n)w_n = f_d(w_n, p, t) \equiv 0 \tag{6}$$

at each time point t_n in the interval $[0, T]$. (Actually, f_d depends on $w_{n-1}, \cdots, w_{n-k}$ because of (5), but this dependence is suppressed here for convenience of notation.) A Newton iteration is used to solve the nonlinear algebraic difference equations so that the overall computational task of time-domain simulation is the following double iteration.



(7)

With only trivial modifications (7) applies to dc and ac cases also. In Section II we show how the routine for inverting F_{dl} may be used also to obtain the performance gradient $\partial\Phi/\partial p$ by the efficient adjoint approach [5]. (With this approach the cost of computing the np-dimensional vector $\partial\Phi/\partial p$ is roughly equal to that involved in (7)). Thus the tableau approach achieves dc, ac, and transient optimization with one matrix operation, namely, the inversion of F_{dl} in (7). Note (7) is essentially a two-step algorithm. The first step is to evaluate f_d and F_{dl}. A virtue of the tableau approach is that f_d and F_{dl} may be read directly from a simple input language using one input card per branch. The second computationally critical step is the inversion of the sparse matrix F_{dl}. This operation is handled by package programs for sparse Gaussian elimination. The tableau format allows these packages to be general-purpose programs not special to the network application. In Section III we describe prediction and control of truncation error in a variable-order variable-time-step application of the differentiation formulas (5) to algebraic–differential systems. This approach is similar to that developed by Gear [8] for strictly differential systems except that we store backward differences instead of the Nordsieck vector. Except for systems with only a small number of differential equations, the use of backward differences is more efficient computationally. This can be established by a simple operations count. Section IV discusses the sparse matrix algorithms, and in Section V we show how the tableau f_d, F_{dl} for the network optimization may be straightforwardly set up from a SCEPTRE-like input language [9].

Sparse matrix packages OPTORD and 1-2-3 GNSO are used to invert F_{dl} (actually the L/U factorization method is employed rather than direct inversion). Section IV gives the specific algorithms employed. The output of OPTORD is a pivoting order, i.e., the order in which the rows and columns of F_{dl} are eliminated in inverting F_{dl}. The ordering selected gives an optimized compromise between 1) minimum roundoff error in the solution Δw, and 2) minimum total operations count in performing the double iteration (7). 1-2-3 GNSO produces the required machine code SOLVE in three partitions: C-SOLVE, to be executed at the beginning of each optimization step, i.e., before entering the time loop of (7); T-SOLVE, to be executed in the time loop of (7); and X-SOLVE, to be executed in the Newton loop of (7). All elimination of variables is done by the Gaussian elimination routines. For example, the program does not require the determination of state variables, trees, nodal admittance matrices, capacitor loops, or any of the various specialized network concepts which tend to complicate a basically simple procedure. The sparse matrix packages OPTORD and 1-2-3 GNSO allow us to dispense with these procedures. In this sense the tableau matrix–Gaussian elimination approach tends to have a simplifying effect on network analysis not only computationally but to some extent theoretically as well. In Section VI we demonstrate that nodal analysis [6] and state-variable analysis [10] may be regarded as special methods of performing Gaussian elimination with different pivoting strategies. From this point of view there are many methods of analysis, and the most appropriate one for purposes of computation is the one which gives the best combination of accuracy and efficiency.

Comparisons of unconstrained OPTORD analysis with OPTORD constrained to eliminate state variables or nodal variables last are among the numerical results given in Section VII. Also, results showing how storage requirements grow with problem size are given for a sequence of examples.

II. The Tableau and Adjoint Sensitivity Computation

The main computation in minimizing $\Phi(p)$ is to compute, given p, $\Phi(p)$ and $\partial\Phi/\partial p$. With these quantities, nonlinear programming algorithms (e.g., Davidon–Fletcher–Powell [11]) can be utilized in minimizing $\Phi(p)$. In this section we present a theorem which demonstrates the role of the tableau matrix F_{dl} in the adjoint-method computation of $\partial\Phi/\partial p$. It is convenient for this purpose to group the algebraic network equations and variables so that

$$a \equiv \text{col}\,(V, i, v)$$

and

$$0 = f(a, q, \Phi, p, t) \equiv \begin{bmatrix} g(a, q, p, t) & \\ Ea \quad -\dot{q} & \\ \phi(a, q, p, t) & -\dot{\Phi} \\ & -\dot{p} \end{bmatrix}, \quad 0 \le t \le T. \tag{8}$$

Thus

$$\Phi = \Phi(T) \equiv \int_0^T \phi(a, q, p, t)\, dt \tag{9}$$

and the format of the tableau matrix F_{dl} is as shown in Fig. 8.

In these terms we may now state, without proof, the following theorem.

Theorem 1

Assume that the system (8) starts at equilibrium, i.e., $\dot{q}(0)=0$, and that T is independent of p, i.e., $\partial T/\partial p=0$. Then

$$\frac{\partial\Phi}{\partial p} = \hat{p}'(0) + \overline{p'} \tag{10}$$

where $\hat{p}'(t)$, $0\le t\le T$, satisfies the adjoint differential equations[2]

$$[\hat{a}', \hat{q}', \hat{\Phi}, \hat{p}'] \begin{bmatrix} \frac{\partial g(t)}{\partial a} & \frac{\partial g(t)}{\partial q} & 0 & \frac{\partial g(t)}{\partial p} \\ E & d/dt & 0 & 0 \\ \frac{\partial \phi(t)}{\partial a} & \frac{\partial \phi(t)}{\partial q} & -d/dt & \frac{\partial \phi(t)}{\partial p} \\ 0 & 0 & 0 & -d/dt \end{bmatrix} = [0, 0, 0, 0] \tag{11}$$

with initial conditions

$$\hat{a}'(T) = \hat{q}'(T) = \hat{p}'(T) = 0, \qquad \hat{\Phi}(T) = -1 \tag{12}$$

[2] $\partial g(t)/\partial a$, etc., denotes $\partial/\partial a\{g(a(t), q(t), p, t)\}$, where $a(t)$, $q(t)$ is the solution of (8).

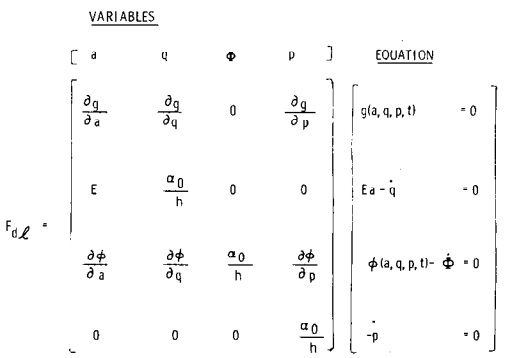


Fig. 8. Algebraic–differential format of tableau.

and $\overline{p'}$ satisfies the adjoint algebraic equations

$$[\overline{a'}, \overline{q'}, \overline{\Phi}, \overline{p'}] \begin{bmatrix} \frac{\partial f_a(0)}{\partial a} & \frac{\partial f_a(0)}{\partial q} & 0 & \frac{\partial f_a(0)}{\partial p} \\ E & 0 & 0 & 0 \\ \frac{\partial \phi(0)}{\partial a} & \frac{\partial \phi(0)}{\partial q} & -1 & \frac{\partial \phi(0)}{\partial p} \\ 0 & 0 & 0 & -1 \end{bmatrix} = [0, \hat{q}'(0), 0, 0]. \tag{13}$$

Note that except for the d/dt entries, the coefficient matrices in (11) and (13) are identical to the tableau operator matrices $F_l(t)$ and $F_l(0)$, respectively. Thus the columns of F_l, operating to the left on the adjoint unknown vector $[\hat{a}', \hat{q}', \hat{\Phi}, \hat{p}']$ give the adjoint sensitivity equations just as the rows of F, (cf., Fig. 4) operating to the right on col $[a', q', \Phi, p']$, give the original differential equations (8). Thus to solve (11) and (13) as well as (8) involves Gaussian elimination applied only to F_{dl} (of Fig. 8), modified as shown in the d/dt entries.

This theorem establishes the actual flow of the computations leading to $\partial\Phi/\partial p$. First (7) is solved yielding $a(t_n)$, $q(t_n)$, $\Phi(t_n)$ at the time points t_n in the interval $[0, T]$. Those components of $a(t_n)$ and $q(t_n)$ which cause F_{dl} to vary are stored on disk. These are returned to core during solution of (11). Since the times t_m involved in solving (11) do not generally coincide with the t_n involved in (7), kth-order interpolation formulas, similar to (5), are used to obtain $a(t_m)$ and $q(t_m)$. The final value of $\hat{q}'(t)$, i.e., $\hat{q}'(0)$, is then passed into the right-hand side of (13) which is solved to give $\overline{p'}$. Finally, $\hat{p}'(0)$ and $\overline{p'}$ are added to give $\partial\Phi/\partial p$.

At this point both $\Phi(p)$ and $\partial\Phi/\partial p$ are available, and an optimization step may be taken. It is important to emphasize that in the iterations involved in solving (7) and (11) only one matrix structure, that of F_{dl}, is subject to sparse Gaussian elimination, and no other matrix operations, e.g., addition, transposition, and multiplication, are necessary.

III. Discretization of the Time Derivative Operator

Two basic assumptions have guided the development of the tableau approach to network simulation: for most networks the tableau matrix 1) is sparse and 2) has widely separate eigenvalues (i.e., it describes a stiff system of ODE). These assumptions dictate that for efficiency, implicit quasi-A-stable integration formulas be used [7]. The back-

ward differentiation formulas (5) of order $k(1 \leq k \leq 6)$ satisfy these requirements. Using the Nordsieck vector [12], Gear [8] has demonstrated their advantage in providing a variable-order variable-time-step approach to minimizing the number of time steps subject to specified allowable integration errors. In the present context of simulation and optimization such an approach is especially necessary because the computation of $\partial\Phi/\partial p$ (cf., discussion of *Theorem 1*) requires storage of all arguments of the nonlinearities at every time step.

Gear's basic strategy, with minor modifications, of step size and order selection based on prediction and control of truncation error have been incorporated into the tableau approach. However, the Nordsieck vector has been replaced with stored backward differences to reduce operations counts when integrating a large number of equations. This replacement gives rise to new difference formulas for prediction and truncation error, and these will be presented along with the description of the discretization of the d/dt operator. Since this operator can act on true charges (q), the objector (Φ), or parametors (p), we state the development in terms of a generic vector x. Thus restated (5) reads

$$-h\dot{x}_{n+1} = \sum_{i=0}^{k} \alpha_i x_{n+1-i} \tag{14}$$

where $x_{n+1-i} \equiv x(t_{n+1-i})$, $h_i \equiv t_{i+1} - t_i$, and $h \equiv h_n$. Note the h_i depend on i in a variable-step method. The coefficients α_i are determined by the requirement that (14) annihilate polynomials of degree $\leq k$. In particular, this means that (14) must be satisfied for each of the $k+1$ substitutions $x_i = (t_{n+1} - t_{n+1-i})^j, j = 0, 1, \cdots, k$, thus yielding $k+1$ equations in the $k+1$ unknowns $\alpha_0, \alpha_1, \cdots, \alpha_k$. These can be solved in closed form, yielding an efficient method for computing the α_i. Since in practice one solves for the vector Δx_i instead of x_i, higher precision is obtained by expressing (14) in terms of the Δx_i. To this end, let

$$x_{n+1-i} \equiv x_{n-i} + \Delta x_{n-i}, \qquad i = 0, 1, \cdots, k$$

and

$$\beta_i \equiv \sum_{j=0}^{i} \alpha_j. \tag{15}$$

Then (14) becomes

$$\sum_{i=0}^{k-1} \beta_i \Delta x_{n-i} + h\dot{x}_{n+1} = 0. \tag{16}$$

Thus in (7) the effect of d/dt operating on q in forming f_d is given by

$$\frac{dq_{n+1}}{dt} = -\frac{\beta_0}{h}(q_{n+1} - q_n) - \frac{1}{h}\sum_{i=1}^{k-1} \beta_i \Delta q_{n-i}. \tag{17}$$

Correspondingly, the d/dt elements in F_l are replaced by $-\beta_0/h$ to obtain F_{dl} (note $\beta_0 = \alpha_0$).

In (7), one has the option of starting the Newton iteration with $w_{n+1}^{(0)} = w_n$, or of predicting from past history an alternative starting value. Experience shows that this prediction step, if efficiently executed, is usually worthwhile. The kth-order prediction formula

$$x_{n+1}^{P} \equiv -\sum_{i=1}^{k+1} \gamma_i x_{n+1-i} \tag{18}$$

has the same order of accuracy as (17). Again, the γ_i are chosen to annihilate polynomials of degree $\leq k$. The coefficients γ_i are also used in computing the truncation error

$$h((\dot{x}_{n+1})_{\text{true}} - (\dot{x}_{n+1})_{\text{approximate}}) \equiv E_k + O(h^{k+2}) \tag{19}$$

where $(\dot{x}_{n+1})_{\text{approximate}}$ denotes the value given by (14). E_k is obtained by expanding the terms x_{n+1-i}, $i = 0, 1, \cdots, k$, in (14) in a Taylor series around t_{n+1}. Using the fact that (18) and (14) annihilate polynomials of degree $\leq k$, one can show that

$$E_k = H_k \frac{h^{k+1}}{k+1} \frac{d^{(k+1)}x_{n+1}}{dt^{k+1}} = -\delta_0\left(x_{n+1} + \sum_{i=1}^{k+1} \gamma_i x_{n+1-i}\right) \tag{20}$$

where

$$H_k \equiv \frac{1}{k!}\sum_{i=0}^{k} \alpha_i \left(\frac{t_{n+1-i} - t_{n+1}}{h}\right)^{k+1} \equiv \frac{1}{k!}\prod_{i=1}^{k} \frac{(t_{n+1} - t_{n+1-i})}{(t_{n+1} - t_n)}$$

and where the extra coefficient δ_0 is determined by requiring the right-hand equation of (20) to be satisfied identically for polynomials of degree $k+1$. As in Gear's work on automatic step size and order control [7], the new time step h and order k are chosen so that h is maximum, consistent with

$$\frac{E_k}{h} \leq \frac{E_T}{T} \tag{21}$$

where T is the total integration time and E_T a specified total absolute error. In practice, to reduce roundoff error, the prediction and truncation error formulas (18) and (20) are rewritten in terms of the Δx_i of (16), just as (17) was obtained from (14).

The *Lemma* presented demonstrates that it is unnecessary to predict any components of the w vector (cf., (7)) other than those which cause the tableau matrix F_{dl} to vary with w. In the *Lemma* u_{n+1} represents the subset of w_{n+1}, on which depend the nonlinear components of the tableau error vector $f(w, t)$. (For the example of Fig. 4, $u = \text{col}\,(v_3, v_2, i_3) = \text{col}\,(w_{12}, w_{11}, w_6)$.)

Lemma

Let w_{n+1}^{P1} be the result of predicting (cf., (18)) all components of w_{n+1}, and w_{n+1}^{P2} be the result of predicting only the subset u_{n+1} of w_{n+1}. Then $w_{n+1}^{P1(1)}$ (obtained from the first Newton iteration step of (7) with $w_{n+1}^{(0)} = w_{n+1}^{P1}$) is equal to $w_{n+1}^{P2(1)}$ (obtained from (7) with $w_{n+1}^{(0)} = w_{n+1}^{P2}$).

Thus in contrast to Gear's method [7], it is not necessary to predict all quantities operated upon by d/dt. Since u is generally a very small subset of w (3 components of 20 in Fig. (4), the computational advantage of predicting u instead of predicting either 1) all the variables, or 2) all the nonlinearities plus all the charges [13] is substantial. The proof of the *Lemma*, which will be omitted here, is based on the fact that the tableau matrix is identical for the two modes of prediction, i.e.,

$$F_{dt}(w_{n+1}^{P1}) \equiv F_{dt}(w_{n+1}^{P2}).$$

In summary, three difference operators are used, one each for $\dot{x}_{n+1}$ (as in (17)), $x_{n+1}^{(0)}$ (as in (18)), and $H_k(h^{k+1}/(k+1)) \cdot (d^{(k+1)}x_{n+1}/dt^{k+1})$ (as in (20)). Each of these formulas use stored backward differences Δx_{n+1-i}. The coefficients α_i, γ_i, δ_0 are computed at each time step. The time step and order (k, $1 \le k \le 6$) are determined as recommended by Gear using the inequality (21). Prediction is only done for the variables u_{n+1} which cause F_{dt} to vary.

IV. Sparse Matrix Methods

There are basically two sparse matrix programs used. The first program OPTORD deals with finding an optimal pivot order, i.e., an order of elimination of rows and columns so that "fill-in" and "operations count" are minimized at the same time numerical stability is retained. Qualitatively, in terms of reducing fill-in, the program OPTORD uses a strategy due to Markowitz [4]. Similar techniques can be found in Tinney–Walker [3] and Dantzig *et al.* [4]. OPTORD is novel, however, in its introduction of variability type to identify and avoid redundant operations in cyclic computations. The use of a numerical test in OPTORD is also not usually incorporated in these methods.

The second program 1-2-3 GNSO takes the permutation generated by OPTORD and, taking into account the variability types of the matrix elements, writes five machine code programs (called C-SOLVE, T-SOLVE, X-SOLVE, B-SOLVE, and A-SOLVE) which execute only the nontrivial arithmetic operations necessary in the Crout factorization algorithm for Gaussian elimination [15]. This program is based on a similar program GNSO [1], which generated only a single FORTRAN program, and did not recognize variability type.

We discuss in turn OPTORD and 1-2-3 GNSO, both of which recognize three basic properties of the nz nonzero elements of the ns × ns matrix S, namely, 1) location I, J; 2) numerical value $S[I, J]$; and 3) variability type $VT[I, J]$. The six variability types are listed in Table I.

TABLE I

Type 1	elements which are $+1$ } called topological type
Type 2	elements which are -1 }
Type 3	elements which never change, called c type
Type 4	elements which change with design parameters, called p type
Type 5	elements which change with time, called t type
Type 6	elements which change with the unknown, called x type

These types are represented in the example of the matrix F_l of Fig. 6. In this example, the 1, 5 and 5, 11 elements of F_l are the topological type ($VT[1, 5]=1$, $VT[5, 11]=2$), the 12, 5 element is p type ($VT[12, 5]=4$), the 17, 17 element is t type (because d/dt is replaced in F_{dt} by $-\alpha_0/h$, which changes with time in a variable-step method, $VT[17, 17]=5$), and element 15, 7 is of x type ($VT[15, 7]=6$). Note that in a time-domain optimization, topological-type elements are invariant, p-type elements change every optimization step, t-type elements every time step, and x-type elements every Newton iteration (cf. (7)).

Ideally, an arithmetic operation is done only when and if it must be done. Thus multiplication of a t-type element by a p-type element should properly be executed in the time loop of (7) but not in the inner Newton loop. To this end there is associated with each multiplication of the term $S[k, L]*S[K, L]$, a variability type

$$VTM \equiv \max\left[VT[k, L], VT[K, L]\right]$$

which identifies when the multiplication must be done.

In order that OPTORD may reduce the total operations count in a given design problem, a set of weights $\mathrm{wt}[i]$, $i=1, \cdots, 6$, are specified, one for each of the six variability types. Thus the weight $\mathrm{wt}[i]$ can be an estimate of relatively how often i-type operations must be executed during an optimization run.

The permutation vectors rp and cp are used to specify the pivoting order. Thus the ith pivot is in row $rp[i] \equiv I$ and column $cp[i] \equiv J$. The task of OPTORD is to select I and J so as to minimize the ith weighted pivot cost

$$PC[I, J] \equiv \mu M[I, J] + \nu R[I, J] \tag{22}$$

where $\mu, \nu \ge 0$ are specified input parameters, M is a weighted multiplication count, and R is a roundoff error factor. We define

$$M[I, J] \equiv \sum_{j=1}^{m_{IJ}} wt_j \tag{23}$$

with wt_j standing for the weight associated with the highest variability type involved in the jth of the m_{IJ} multiplications required in eliminating row I and column J. Also,

$$R[I, J] \equiv \frac{1}{r_I} \frac{\sum_j |S[I, j]|}{|S[I, J]|} \tag{24}$$

where j runs over the r_I nonzero column indices in row I.

The ith of the n formally identical steps in the process of selecting rp and cp is described by the following algorithm.

OPTORD Algorithm

At the end of the $(i-1)$ step, $i-1$ eliminations have been carried out. Thus in the ith step the $(n+1-i) \times (n+1-i)$ matrix $S^{(i)}$ is to be considered (note $S^{(1)} \equiv S$) in which the nonzero locations, the values, and the variability types of its elements are known.

1) For each row K determine the subset $l[j]$, $j=1, \cdots, \hat{r}_K$ of the r_K nonzero columns of the Kth row of $S^{(i)}$ for which

$$TOL \cdot R[K, l[j]] \le 1 \tag{25}$$

where $0 \le TOL \le 1$ is a threshold factor. The remaining $r_K - \hat{r}_K$ elements in row K have numerical values which are considered too small to be candidate pivots.

2) Determine the $\hat{r}_K$ values of $M[K, l[j]]$ (cf., (23)), $j=1, \hat{r}_K$, associated with the elimination of row K and column $l[j]$ (element $K, l[j]$ is the candidate pivot). In Gaussian

elimination a multiple of the pivot row K is added to each row k in $S^{(i)}$, which has a nonzero element in column $l[j]$, in such a way that the $k, l[j]$ element becomes zero. Thus for each such row k we compute

$$S^{(i)}[k, L] - S^{(i)}[k, l[j]] * (S^{(i)}[K, L]) / S^{(i)}[K, l[j]]) \quad (26)$$

where L runs over the union of column indices in rows k and K. Associate with each nontrivial multiplication (represented by $*$ in (26)) its variability type

$$VTM[k, L] \equiv \begin{cases} \max\{VT[k, l[j]], VT[K, l[j]], VT[K, L]\} \\ 0, \quad \text{if } S^{(i)}[K, L] = 0. \end{cases} \quad (27)$$

If there are c_j nonzero elements in the pivot column $l[j]$ of $S^{(i)}$, then $(c_j - 1)(r_K - 1)$ such multiplications would be required to use element $K, l[j]$ as the next pivot. The weighted multiplication count for each of the $\hat{r}_K$ pivots $K, l[j]$, $j = 1, 2, \cdots, \hat{r}_K$, is therefore given by[3]

$$M[K, l[j]] = wt[VT[K, l[j]]] + \sum_k \left(\sum_L wt[VTM[k, L]] \right) \quad (28)$$

where k runs over the nonzero row indices in column $l[j]$, and L runs over the union of column indices in rows k and K. The first term accounts for the computation of $(S^{(i)}[K, l[j]])^{-1}$ and for $k = K$; the sum accounts for the multiplications $(S^{(i)}[K, l[j]])^{-1} * S^{(i)}[K, L]$.

3) Compute $PC[K, l[j]]$, $j = 1, \cdots, \hat{r}_K$, using (24) and find the minimum pivot cost column $JC[K]$ for each row K, $K = 1, 2, \cdots, n + 1 - i$, i.e.,

$$PC[K, JC[K]] = \min_j \{PC[K, l[j]]\}. \quad (29)$$

4) Find the minimum operations-count row I, i.e.,

$$M[I, JC[I]] = \min_K \{M[K, JC[K]]\}. \quad (30)$$

5) Execute the numerical Gaussian elimination of row $I \equiv rp(i)$ and column $J = JC(I) \equiv cp(i)$, creating the $(n-i) \times (n-i)$ matrix $S^{(i+1)}$.

6) Update the variability types of each nonzero element in $S^{(i+1)}$ (cf., (26)), according to

$$VT[k, L] = \max\{VTM[K, L], VT[k, L]\}.$$

Note that if the operation count weight is set to zero (i.e., $\mu = 0$ in (22)), step 3 selects the largest pivot element in row K. Thus partial pivoting [17] is effected in each row, and the row is chosen (cf., (30)) to minimize the weighted multiplication count. The decision to have OPTORD minimize only the number of multiplications required at each step of Gaussian elimination was made on the basis of some comparison with other methods. Roughly, the result of this comparison was that it was important to base the next pivot choice on the result of executing the previous pivot. However, how one then chose the best pivot was not so critical and many methods seemed to give the same benefits. These conclusions are in agreement with those reported in [15]. The choice of minimizing multiplications was in keeping with minimizing the length of the SOLVE code and maximizing execution speed.

[3] We define $wt[0] = 0$ since VTM can be zero (cf., (27)).

1-2-3 GNSO

1-2-3 GNSO creates an executable nonlooping program for solving both $b = Sx \equiv LUx \equiv Ly$ and $\hat{b}' = \hat{x}'S \equiv \hat{x}'LU \equiv \hat{y}'U$. 1-2-3 GNSO differs from GNSO [1] mainly in its use of variability type in creating five distinct lists of machine instructions: C-SOLVE; T-SOLVE; X-SOLVE; B-SOLVE; and A-SOLVE (GNSO creates a single list of FORTRAN instructions). C-SOLVE, T-SOLVE, and X-SOLVE compute the factorization of S into the product of upper and lower triangular matrices L and U. C-SOLVE, T-SOLVE, and X-SOLVE need be executed only once per optimization step, time step, and Newton step, respectively (cf. (7)). B-SOLVE carries out the back substitutions $Ly = b$, $Ux = y$ required in simulation (cf., (7)). A-SOLVE carries out the adjoint back substitution $\hat{b}' = \hat{y}'U$, $\hat{y}' = \hat{x}'L$ required in the sensitivity computation (11) or (13).

The logic of 1-2-3 GNSO is based on the Crout method of factorizing S into LU. For purposes of presentation here it will be assumed that S has been rearranged so that the pivots $S[rp[K], cp[K]]$ appear on the diagonal of S. Thus the Crout formulas are for $K = 1, 2, \cdots, n$,

$$L_{IK} = \left(S_{IK} - \sum_{J=1}^{K-1} L_{IJ} U_{JK} \right), \qquad I = K, \cdots, n$$

$$U_{KI} = \left(S_{KI} - \sum_{J=1}^{K-1} L_{KJ} U_{JI} \right) \div L_{KK}, \qquad I = K + 1, \cdots, n. \quad (31)$$

(Note $U_{KK} \equiv 1$.) The index ordering indicates how the Kth column of L and Kth row of U are to be computed sequently and assures that all right-hand-side terms of (31) are known.

As in OPTORD the variability type of each new L_{IK} or U_{KI} term is equated to the highest type of any term in (31). Similarly, each product in (31) is given the highest type of its two factors. These variability types are used to decide to which lists, C-SOLVE, T-SOLVE, or X-SOLVE, the instructions created to compute each L_{IK} (or U_{KI}) are to be added. For a multiplication, this is done by placing the required machine instructions (LOAD, MULTIPLY, ADD) at the end of the appropriate list. For example, instructions for x-type multiplications are added to X-SOLVE. If a multiplication involves a topological type factor (± 1), the LOAD and MULTIPLY instructions are suppressed. If a number of topological type terms are to be added, the result must be topological (± 1 or 0) (zero if the number of such terms is even).[4] In this

[4] That the sum of two or more terms of topological type in a Crout formula is also of topological type (± 1 or 0) is due to the fact that the tableau with all nontopological type elements replaced by zero is a totally unimodular matrix [18]. This is because in the tableau we have made the restriction that not more than one topological element may appear in a branch-constitutive-relation row. Therefore, since the incidence matrix A is totally unimodular and any totally unimodular matrix augmented by rows or columns of the identity matrix is totally unimodular, it follows that the topological part of the tableau matrix is also totally unimodular. The result then follows from the fact that the elements of L and U can be expressed in terms of subdeterminants of the tableau matrix.

case the only instruction generated is a LOAD of ± 1 (or no instruction if the result is 0).

If a given formula (31) involves multiplications of various types, we have

$$L_{IK} \text{ (or } U_{KI}) \equiv TC(IC) + TT(IT) + TX \qquad (32)$$

where $TC(IC)$ is a temporary storage location for the sum of terms of ± 1 p type or c type and $TT(IT)$ a temporary location for the sum of t-type terms. TX represents the sum of the remaining terms, but no temporary storage location is required. The instructions required for computing $TC(IC)$, $TT(IT)$, and TX are placed at the end of the C-SOLVE, T-SOLVE, and X-SOLVE lists, respectively. The detailed procedure for $TT[IT]$ (the procedure is similar for TC and TX) is given in the following list. We add the following instructions to the T-SOLVE list.

1) LOAD, ADD S_{IK} (if S_{IK} is t type and nonzero)
2) LOAD, MULTIPLY, ADD (for each t-type product in (31))
3) ADD $TC(IT)$ (if present in (32))
4) STORE L_{IK} (if TX absent)
5) STORE $TT(IT)$ (if TX present).

A development similar to that based on (31) and (32) describes the creation of the instruction lists for the back solve programs B-SOLVE (for solving $Sx=b$) and A-SOLVE (for solving $\hat{x}'S=\hat{b}'$). Like X-SOLVE, B-SOLVE (or A-SOLVE) is executed within the Newton iteration since the right-hand-side vector b and the auxiliary vector y change whenever the unknown variable x changes.

Thus by creating the five lists of instructions to be executed in the appropriate loops of the computation, 1-2-3 GNSO achieves near-ultimate efficiency: multiplications are performed only if they are absolutely necessary.

V. An Input Language for the Tableau Approach

An automated network design program (NDP) has been written, incorporating the approach to automated network optimization previously described. NDP interfaces the sparse matrix, implicit integration, and adjoint sensitivity routines through a simple SCEPTRE-like [9] input language. As described in the following paragraphs, the input language processor passes a complete description of the tableau matrix F_l, which is identified as the S matrix of OPTORD and 1-2-3 GNSO.1-2-3 GNSO then produces a machine code program, SOLVE (cf., Section IV). To conserve storage, the SOLVE code is designed to work only on the nonzero nontopological entries of F_l.

Before describing the input language in detail, the threaded list structure used to compact F_l is described. A vector s is used to store the nonzero nontopological values in the (row-wise) order in which they appear in F_l. A vector SP, which points to s, and a column index vector CI are associated with the nonzero elements of F_l. Finally, a row pointer vector RP is used to identify which column indices belong to a given row. For the example of Figs. 1–5, the array s and the components of SP, CI, and RP, belonging to rows 11–17 of F_l (cf., Fig. 5), are

$$s' \equiv \left(1, -1, L, i_2, \frac{\partial L}{\partial p_1}, \frac{\partial \psi}{\partial v_2}, \frac{\partial \psi}{\partial v_3}, \frac{\partial \phi}{\partial i_3}, \frac{\partial \phi}{\partial p_1}, d/dt\right)$$

$$SP(21\text{–}36) = (1,\ 3,\ 2,\ 4,\ 5,\ 6,\ 2,\ 1,\ 2,\ 7,\ 2,\ 8,\ 1,\ 2,\ 1,\ 9)$$

$$CI(21\text{–}36) = (11,\ 6,\ 17,\ 20,\ 12,\ 13,\ 18,\ 14,\ 19,\ 7,\ 9,\ 20,\ 16,\ 20,\ 12,\ 17)$$

$$RP(11\text{–}17) = (21,\ 22,\ 25,\ 28,\ 30,\ 33,\ 35,\ ). \qquad (33)$$

The row pointers point to the first nonzero column index in a given row. Thus RP and CI combine to define a unique I, J location in F_l for each component of the pointer SP. This pointer in turn identifies the component of s to be associated with $(F_l)_{IJ}$.

1-2-3 GNSO requires knowledge of SP, CI, and RP, as well as a similarly compacted variability-type array VT. Nevertheless, the SOLVE code generated refers only to the compacted s array (which thus replaces the S elements in (31)). Thus, although in the present example, location, value, and variability type must be known for 41 of the 400 elements in F_l, only 10 quantities need be in core during the simulation–optimization phase of NDP. Note also (cf. Fig. 4) that in (33) only 4 quantities, v_3, i_3, v_2, and i_2, need be stored on disk for the purpose of recreating s (i.e., F_l) for the backward time adjoint integration (11).

In the NDP language the input description of the time-domain network optimization problem of Figs. 1–5 is shown in Table II. The first data card defines the reference node (to be node 0 of Fig. 3). The next nb cards, one for each branch in the network, are free format with successive data fields giving the branch label, the start node, the finish node, the element type, and the name and argument list of a user defined function subprogram. The fifth card gives the name of the subprogram which defines the objective function to appear as integrand in (1) and signals NDP to add the objector and parametor to the network. The sixth and seventh cards state that there is one designable parameter and that in the constrained optimization problem (1) the parameter labeled 1 has a lower bound of 0.01, a starting value of 1, and an upper bound of 100. The eighth card identifies the unknown variables required to recreate the matrix F_l in (11). According to the *Lemma* of Section III, those are also the variables which are predicted in the forward time integration (7). Note NDP translates bracketed function arguments into unknown vector components. In the example, $/T2/\rightarrow i_2\rightarrow x(6)$, $/A3/\rightarrow v_3\rightarrow x(13)$, and $/P1/\rightarrow p_1\rightarrow x(20)$. Other examples of notation are $N1$ (voltage of node 1) and $Q3$ (charge on the capacitor in branch 3).

Given the card input of Table II, NDP creates a subroutine VARQ and input to two permanent subroutines

TABLE II

```
NETWORK DESCRIPTION, NREF = 0
1, 1-0,  3 = E(t)
2, 1-2, 12 = QL(/T2/P1/t)
3, 2-0, 11 = PSI (/A2/A3/)
OPTIMIZE PHI (/T3/P1/)
DESIGN PARAMETERS, NP = 1
P1 (0.01, 1, 100)
NONLINEARITIES /A3/A2/T3/T2/
```

TABLE III

```
f(11) = x(11) - FTIME(t)
f(12) = QL(x(6), s(3), x(20), s(4), t) - x(17)
f(13) = PSI(x(12), s(5), x(13), s(6))
f(15) = PHI(x(7), s(7), x(20), s(8)) - x(9)
```

TABLE IV

```
FUNCTION QL(X1, S1, X2, S2, T)
S1 = FL(X2, S3, T)
QL = X1 * S1                    (≡i2*L)
S2 = X1 * S3                    (≡i2*∂L/∂p1)
RETURN
END
```

FTOP and QTVARQ. These subroutines compute the tableau error vector f and the compacted version s of the tableau matrix F_t. The input to FTOP is the (suitably compacted) incidence matrix A. Note there is a $+1$ (-1) in the column of A corresponding to the start (finish) node. Thus A is essentially read directly from the input cards. The result of FTOP is the evaluation of the first $nn+nb$ components of f.

The generated subprogram VARQ for the present example comprises the list of FORTRAN instructions shown in Table III. There is a one-to-one correspondence between these FORTRAN instructions and input cards 2–5 of Table II, except that the argument lists in the user-defined function subprograms, QL, PSI, and PHI, differ from those on the input cards. This difference is for user convenience. The convention is that given the unknown vector x the subprograms must return not only the function value (e.g., QL) needed for computing f, but also the partial derivatives (needed for evaluating F_t) of the function value with respect to members of the argument list which are components of the unknown vector x, e.g., $s(3) \equiv \partial QL/\partial x(6) \equiv L$, $s(4) \equiv \partial QL/\partial x(20) \equiv i_2(\partial L/\partial p_1)$. Thus unknown vector components in the argument list are always followed by a component of the compacted tableau matrix array s. For user convenience these extra arguments are left out of the input cards.

The user-specified FORTRAN function subprogram QL is described as an example in Table IV. In this function subprogram, another FORTRAN function subprogram FL is called to evaluate the inductance $L \equiv FL$ (dummy argument $S1$) as a function of the time T and the design parameter p_1 (dummy argument $X2$). The value of L is stored in $S(3)$ (cf., Table III). The magnetic flux for this linear time-varying inductor is then computed and stored in QL. Then the derivative $\partial QL/\partial p_1$ (dummy argument $S2$) is computed in terms of $\partial L/\partial p_1$ (dummy argument $S3$) and stored in $S(4)$.

The third executable program QTVARQ creates the $Ea\text{-}\dot{q}$ (cf. (8)) rows of the tableau vector f and the d/dt elements of the tableau matrix. QTVARQ computes $Ea\text{-}\dot{q}$ by replacing $\dot{q}$ by a summation as described by (17).

Note that FTOP, VARQ, and QTVARQ operate directly on the input data of Table II with a minimum of intervening data processing. There are no instructions in NDP which refer to excess or link capacitors or tree inductors. Thus the tableau approach imposes no topological restrictions whatsoever on the type or connectivity of the network elements.

Note also that functions like those in Table IV are entirely user defined. Any function of any group of variables may thus be defined by the user so that there is no restriction on the type of elements treatable by the tableau approach.

VI. Relation Between Pivoting Order and Network Methods of Analysis

When the time discretization procedure (cf., Section III) is carried out before any reduction is made on the network equations, the only constraint on pivoting order is that the numerical value of the pivot element should not be too small. From the point of view of pivot order selection, we examine two of the principal methods of network analysis: nodal analysis and state-variable analysis. For ease of illustration consider the tableau of Fig. 9 which describes a network of only capacitances and other admittance-type elements but no forcing terms. Using the pivoting order (4, 6), (2, 4), (3, 5), (5, 3), (6, 2) we obtain the nodal equations

$$\left\{A_2' Y A_2 + A_1' C A_1 \frac{d}{dt}\right\} V = 0 \tag{34}$$

where the matrix operating on node voltages V is known as the nodal admittance matrix.

The state equations are more difficult to describe in terms of a pivoting order since it requires finer groupings to be made on the variables. Thus a proper tree must be obtained,[5] the tree variables separated from the link variables, and all variables except the capacitor voltages in the proper tree and the inductor currents in its links eliminated. For a simple example, it is now assumed that the capacitor branches of Fig. 9 form a tree; hence this is a proper tree. Thus for the tableau of Fig. 9, the pivoting order (2, 1),

[5] To illustrate that in the general case the state-variable method corresponds to a particular set of pivots and this set gives rise to a proper tree, consider the following constraints on the pivot order.

1) Select the first nb (number of branches) pivots from $AV - Iv = 0$ rows of F_{dt} (Fig. 6). If possible, let the kth pivot fall in the first nonzero column in the kth row of A. In this case, which will occur nn (number of independent nodes) times, the kth branch is in the tree. If after $k-1$ pivots, the kth row of A has been annihilated, then pivot on $I_{kk} = (F_{dt})_{nn+k,nn+nb+k}$. In this case the kth branch is a link.

2) Select the next nn pivots to fall in locations in the $A'i = 0$ rows which are the transpose of the nn pivot locations in A.

These constraints cause a tree to be selected whose identity depends on the order of branches on the input cards. A proper tree [10] can be obtained by rearranging the tableau so that voltage sources come first in the list of branches followed by capacitors, conductors, resistors, inductors, and current sources.

V	i_1	i_2	v_1	v_2	q
	A_1^T	A_2^T			
A_1			-1		
A_2				-1	
			C		-1
		-1		Y	
	1				$-\frac{d}{dt}$

Fig. 9. Simplified tableau.

(3, 5),(1, 2), (4, 6), (5, 3) yields the equations

$$C\frac{dv_1}{dt} = -(A_1')^{-1}A_2' Y A_2 A_1^{-1} v_1 \tag{35}$$

which are the state equations in this case.

In both nodal and state-variable analysis the equations are reduced to their standard form and then the d/dt operator is replaced by a difference operator by some procedure analogous to that described in Section III. The difference equations thus obtained are identical to the difference equations obtained by first replacing the d/dt operator and then pivoting in the same sequence as was done to obtain the state-variable or nodal-variable equations.

Both cases (34) and (35) represent a reduced form of the original tableau matrix F_t. Thus the classical network methods may be regarded as special cases of the tableau approach. From this point of view the particular methods are identified by the constraints they impose on the pivoting order in Gaussian elimination. Since these constraints on the pivoting order are not in general consistent with objectives of minimizing operations count, it follows that these classical network methods, being primarily motivated by analytical simplicity, are intrinsically slower than the sparse tableau approach.[6] The sparse tableau approach also has an advantage in terms of the possibility for higher precision answers. To illustrate this, consider the following system of linear equations

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}. \tag{36}$$

Let this represent the unreduced tableau and the system

$$(D - CA^{-1}B)y = b_2 - CA^{-1}b_1 \tag{37}$$

represent the reduced equations, e.g., these would correspond to state-variable or nodal analysis. Since the unreduced tableau is taken directly from the input data, the matrices A, B, C, D and vectors b_1, b_2 can be assumed to be precise. The matrix $S \equiv D - CA^{-1}B$ and the the vector $\tilde{b}_2 \equiv b_2 - CA^{-1}b_1$ are the result of some arithmetic operations on the input data and therefore are not precise. Let E be the error in S and ε be the error in $\tilde{b}_2$. Thus in the machine the reduced problem is represented by

$$(S + E)y = \tilde{b}_2 + \varepsilon. \tag{38}$$

To solve (36) or (38) precisely (to the accuracy of the machine) one can use iterative refinement [17]. For (36) this means the iteration

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}\begin{pmatrix} \Delta x^{(k)} \\ \Delta y^{(k)} \end{pmatrix} = \begin{pmatrix} b_1 & -Ax^{(k)} & -By^{(k)} \\ b_2 & -Cx^{(k)} & -Dy^{(k)} \end{pmatrix}$$

$$x^{(k+1)} = x^{(k)} + \Delta x^{(k)}$$

$$y^{(k+1)} = y^{(k)} + \Delta y^{(k)}.$$

For (38) this iteration is

$$(S + E)\Delta y^{(k)} = \tilde{b}_2 + \varepsilon - (S + E)y^{(k)}$$

$$y^{(k+1)} = y^{(k)} + \Delta y^{(k)}.$$

According to Wilkinson [17], in the iterative refinement of the problem $Mz = b$, if the matrix M being inverted can be factored into LU, where $LU = M(I + F)$, with

$$\|F\|_\infty \leq \tfrac{1}{2} \tag{39}$$

and if the error vector $b - Mz^{(k)}$ is computed with accumulation, then $\|z^{(k)} - z^*\|_\infty < 2^{-t}$ for k large enough. (Here z^* is the true solution and t is the number of digits in the machine.) Thus to within the accuracy of the machine and subject to (39), equations (36) and (38) can be solved exactly. In the case of equation (36) we then have an accurate answer. However, in the case of (38) we only have an accurate answer to equation (38) but not necessarily equation (36) which was our original purpose. Thus the initial errors E and ε made in manipulating the input data to form the reduced equations cannot be recovered.

VII. Numerical Results for a Practical Example

The current switch–emitter follower logic chain of Fig. 10 is used to demonstrate optimal design and large-scale application capabilities of the tableau approach and to compare its computational efficiency with competitive methods. Each five-terminal stored model in Fig. 10(c) represents one logic stage (Fig. 10(b)) in the chain. Each of these logic circuit models use three transistors, which in turn are represented by the high-speed transistor model of Fig. 10(a).

The automated design problem selected has eight logic stages and is characterized by the parameter set $p' \equiv (R_1, R_2, R_3)$ (for the lower and upper parameter bounds we use $p_m' \equiv (0.05, 0.05, 0.05)$ and $p_M' \equiv (0.15, 0.15, 0.15)$). The design objective is to realize a specified output voltage waveform $V_{out}(t)$ for a given input voltage waveform $V_{in}(t)$ (Fig. 11). The specified output is characterized by delay, rise time, and up and down levels. Thus the objective instantaneous function $\phi = \frac{1}{2}(V_{realized} - V_{specified})^2$ vanishes when the shaded area of Fig. 11 vanishes. For an eight-circuit logic chain the result of six Fletcher–Powell optimization steps (bottom of Fig. 11) is that $\Phi(p)$ decreases from 0.7 to 0.0003.

[6] It can be seen that the symbolic elimination of topological equations is formally equivalent to the 1-2-3 GNSO process of converting multiplications by ± 1 into additions or subtractions.

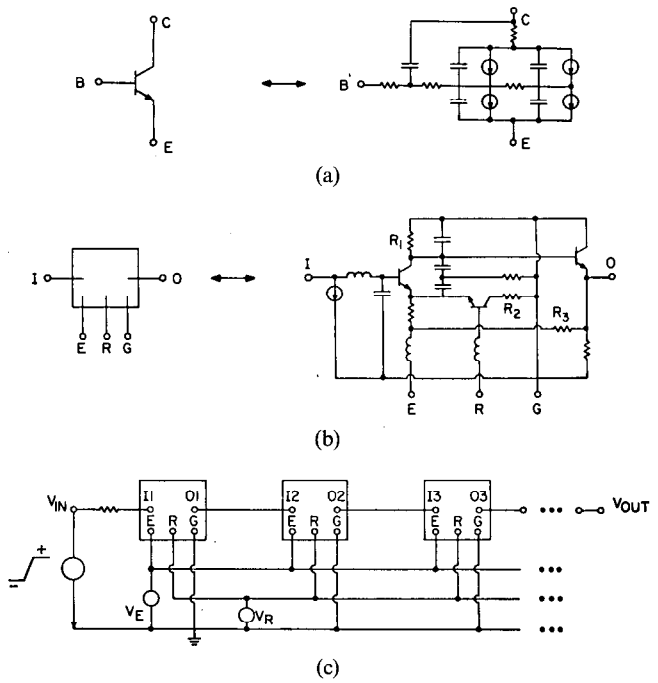
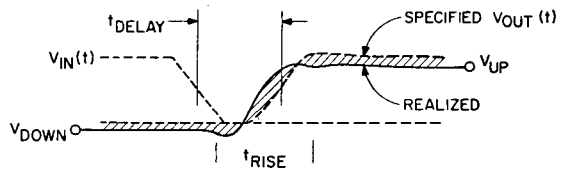


Fig. 10. Current switch–emitter follower circuit logic chain. (a) Two-dimensional transistor model. (b) Current switch–emitter follower logic stage. (c) Logic chain.



	Φ(p)	R1	R2	R3
INITIAL	.7	75Ω	100Ω	150Ω
FINAL	$.3 \times 10^{-3}$	114Ω	100Ω	111Ω

Fig.11. Three-parameter optimization of eight-stage logic chain.

TABLE V

Problem complexity (8 circuit optimization)	
nodes	180
branches	460
nonlinear elements	112
differential equations	192
total equations	1300
Optimization (Fletcher–Powell with quadratic search)	
designable parameters	3
gradient computations ($\partial\Phi/\partial p$)	6
function evaluations ($\Phi(p)$)	36
time steps	30 000
newton iterations	85 000

Various aspects of problem complexity are given in Table 4. Note that for this example, p-type elements (variability type = 4) of the tableau matrix F_l varied 36 times and t- and x-type elements varied 30 000 and 85 000 times, respectively.

To show how code length varies with problem size, we give in Table V the results of applying OPTORD/1-2-3 GNSO to logic chains of 1, 8, and 22 circuits.

For the data given in Table VI we made no attempt to try to minimize the length of the code but chose partial pivoting ($\mu=0$, $v=1$ in (22)) to insure accurate answers. The data indicate that for the one-circuit case, F_l has 462 nonzero elements, of which 92 are stored in the threaded list (33). The L/U factorization adds another 450 fill-ins so that there are 912 nonzero elements in L/U, of which 360 must be stored during execution of the generated SOLVE code. The sum of the lengths of the data, and of the C-, T-, X-, B-, and A-SOLVE codes is 27 316 bytes. The central processing unit IBM Model 91 times required for function evaluation (time-domain simulation of 24 ODE) and gradient computation are 5 s and 3 s, respectively. Note that the various data in Table VI (in particular note the total code length) increase approximately linearly with the number of circuits analyzed and, therefore, with the order of F_l. (Such linear growth may be typical for network tableau matrices.) However, the growth of simulation time ($\Phi(p)$ column of Table VI) is supralinear. This is because both the number of time steps and the number of Newton iterations per time step generally increase with problem size.

The results of Table VII illustrate the effects of varying the ordering strategy to minimize the length of x-type code (X-SOLVE and B-SOLVE). The data are for the one-circuit case of Fig. 10(b) (this time without objectors and parametors so the original tableau has only 446 nonzero elements) and serve to quantify the comparison of the tableau approach using unconstrained OPTORD with OPTORD constrained to correspond to competitive network analysis methods given in Section VI. In each case the ordering parameters TOL, μ, v, and wt[i], $i=1, 2, \cdots, 6$, (cf., (22)–(26)) were chosen by trial and error to minimize code length irrespective of accuracy considerations and are tabulated in Table VIII.

For the nodal-analysis method, the last nn (number of independent nodes) pivots were constrained to come from the upper-left-hand partition of F_l (originally zero in Figs. 6 and 9). For the state-variable method, the last ns (number of state variables is $ns=21$) pivots were constrained to be in the state-variable columns and in the corresponding d/dt rows. Within these constraints, the full OPTORD algorithm was used to select the pivots.

One basis for comparing the performance of the generated codes is the sum of the X-SOLVE and B-SOLVE code lengths[7] since these are executed in the innermost loop of the program (85 000 times in the example of Table V). In Table VII the data (compare cases 1 and 2 to 5 and 6) demonstrate our general experience that the pivot order constraint imposed by the nodal-analysis method did not significantly increase SOLVE code lengths over unconstrained OPTORD. The pivot order constraint (cases 3 and 4) imposed by the state-variable analysis method caused a more substantial increase in code length. In all cases observed to date, unconstrained OPTORD has given the best performance, with nodal analysis a fairly close second. (An explanation of this phenomenon will be given in a subsequent paper.)

[7] Since the generated codes are relatively homogeneous and have no loops, the code length is roughly proportional to execution time.

TABLE VI

PARTIAL PIVOTING ($\mu=0$, $\upsilon=1$) OPTORD/1-2-3 GNSO FOR LOGIC CHAIN OF 1 CIRCUIT, 8 CIRCUITS, 22 CIRCUITS (PARAMETORS AND OBJECTOR INCLUDED)

Nodes/Branches	F_l		L/U		Total Code in Bytes	Model 91 Time (s)		Number of ODE
	Nonzero	Stored	Fill	Stored		$\Phi(p)$	$\partial\Phi/\partial p$	
16/61	462	92	450	360	27 316	5	3	24
180/460	3472	680	3735	3430	276 404	120	30	192
492/1260	8632	1736	9332	8617	713 504	900	500	928

TABLE VII

COMPARISON OF SPARSE TABLEAU APPROACH (NDP) WITH CLASSICAL NETWORK METHODS

Case	Analysis Mode	OPTORD/1-2-3 GNSO Output				
		L/U		Code Length (bytes)[a]		
		Fill	Stored	*T-SOL*	*X-SOL*	*B-SOL*
1	Nodal -OPTORD	215	348	{2200 / 1564}	{1868 / 1656}	{7864 / 6754}
2	Nodal -OPTORD	194	390	{2812 / 1790}	{2388 / 1932}	{7668 / 6786}
3	$S-V$ -OPTORD	472	426	{4776 / 3968}	{2828 / 2630}	{9584 / 8288}
4	$S-V$ -OPTORD	500	580	{4044 / 3150}	{14 492 / 14 102}	{10 620 / 9 414}
5	NDP -OPTORD	191	344	{2576 / 1652}	{2048 / 1686}	{7632 / 6554}
6	NDP -OPTORD	204	329	{2820 / 1956}	{1572 / 1400}	{7792 / 6616}

[a] Upper number in bracketed pairs represents output of current 1-2-3 GNSO compiler. Lower number represents estimates based on hypothetical optimum compiler.

Note in Table VII a correlation between generated code lengths and the number of stored quantities in L/U. This suggests that minimizing the fill in of non ± 1 quantities may be the best criterion for optimizing code lengths.

When circuits become much larger, as in the example of Table V, accuracy requirements rather than code length can become the dominating factor in selecting ordering parameters. The nodal and state-variable column constraints on the pivoting order prevent the implementation of a true partial pivoting strategy ($\mu=0$, $\upsilon=1$) and thus contribute to a reduction in the accuracy of the answers. This consideration may be the cause of problems observed in NODAL OPTORD code for large values of the time step h and with S-V OPTORD for small values of h. Another accuracy consideration is the fact that the number of Newton iterations per time step increases with problem size for large networks. The evidence suggests that the Newton iteration is not only solving the nonlinear difference equations in (7), but also serving to refine the solutions of the linear equations. This consideration has thus far prevented successful runs on very large networks with parameters set for other than partial pivoting. This aspect of the accuracy problem may also prevent extensive exploitation of possible sparsity of the right-hand-side vector. One might think that for linear time invariant rows of the tableau vector f (e.g., the first 11 rows in Fig. (4), the right-hand side would stay zero if started at zero. This is not the case. Instead it has been found necessary to evaluate the full right-hand side at every Newton iteration to obtain rapid convergence.

TABLE VIII

ORDERING PARAMETERS FOR CASES OF TABLE VII

Case	*TOL*	OPTORD/1-2-3 GNSO Input							
		μ	υ	*wt*(1)	*wt*(2)	*wt*(3)	*wt*(4)	*wt*(5)	*wt*(6)
1	0	0.5	1	100	100	500	400	450	1500
2	0	1	0	100	100	150	200	150	300
3	0	0.5	1	100	100	500	400	450	1500
4	0	1	1	100	100	500	600	450	900
5	0	1	0.5	100	100	90	101	90	150
6	0	1	0	100	100	102	101	102	150

The preceding results and discussion, combined with the error discussion of Section VI, suggest that nodal or state-variable constraints on the pivoting order offer no appreciable storage or execution speed advantages and create the possibility of accuracy problems.

VIII. CONCLUSIONS

The advantages claimed for the tableau approach are 1) generality, 2) simplicity, 3) speed, and 4) problem size.

Generality is claimed because any system of differential and algebraic equations can be handled. In particular, for electrical networks no topological restrictions are necessary and there are no restrictions on the type of elements allowed. This allows, for example, the inclusion of the objective function in the tableau by introducing the objector element.

Simplicity is achieved with the tableau approach by basing the main computational task of the program on Gaussian elimination. All other computations have a simple direct relation to the input data for the program. Although the programs OPTORD and 1-2-3 GNSO are not simple, they are general in nature and relate to the more general problem of solving $f(x, \alpha)=0$ by Newton's method for different values α where $\partial f/\partial x$ is sparse and has elements with different variability types. It therefore seems wise to develop such programs to a high degree and base more special programs (such as network design programs) around these developments.

The claim of speed is based on the fact that operations count in the basic Newton loop has been minimized so that only necessary operations are executed. This has been done by using variability type and near-optimal ordering algorithms in OPTORD. In addition, the number of time steps

required for integration has been made small by using implicit methods and Gear's error control algorithms.

Large problems can be handled under this program because storage is saved by not storing 0's and 1's. In addition, OPTORD saves space by causing little fill in. Since the SOLVE code contains no loops, it can be stored in auxiliary storage and executed as it is brought into core thus requiring no storage. (However, for maximum speed, it should remain in core.)

As was shown in Section VII, the S-V OPTORD and NODAL-OPTORD methods offer no advantages over the unconstrained OPTORD methods. In fact, possibly serious accuracy problems may occur. Note that the NODAL-OPTORD column constraints cause differencing errors in the back substitution which may be serious when arguments of nonlinearities are branch voltages. Thus if such a branch voltage is small in comparison with its terminal node voltages, it will be computed to fewer significant digits.

Note that some speed and storage advantages may accrue to the nodal or state-variable methods if the elimination leading to the appropriate final variable set is done symbolically at the input processor level. Since variables which enter linearly and are neither energy storage nor involved in output can be eliminated, the amount of B-SOLVE code can be reduced. However, the excess B-SOLVE code (which corresponds roughly to solving $Ax=(b_1-By)$ in (36)) could, with some programming effort, be removed from the output of 1-2-3 GNSO. Thus this excess code is a deficiency of the implementation but not of the sparse tableau approach. Note that a priori symbolic reduction introduces topological matrix–vector operations and complicates the forming of the Jacobian matrix and right-hand side vector. (These additional operations correspond roughly to the forming of $D-AC^{-1}B$ and $b_2-CA^{-1}b_1$ in equation (37).) In contrast, the analogous operations for OPTORD analysis are executed by the highly efficient SOLVE code.

In NDP we have chosen to emphasize speed at some cost to problem size by keeping the SOLVE code in core. On a large machine such as 360/91 with 2×10^6 bytes of memory, this turns out to be a good choice. On this machine the capacity limitation of NDP seems to be about 2500 branches. A transient analysis of a network with 1200 branches took about 20 min. An entire optimization involving three parameters of a 460 branch network and requiring 36 function evaluations and 6 gradient evaluations took 1 h. Extrapolating linearly, we calculate that optimization for a 2500 branch network would require 7 h. However, this is low since larger networks usually require more time steps per transient integration and more Newton iterations per time step. Further, for a large circuit more than three parameters will normally be used which surely will increase the number of function and gradient evaluations required. Thus all indications are that 7 h is a very rough lower bound for optimization of a 2500 branch network. It would not be surprising if as many as 20–60 h were required. On the other hand, no serious attempt has been made yet to optimize the various parameter settings in NDP to cut down the number of time steps, Newton iterations, etc., so that this figure may be cut down.

If transient analyses for even larger networks are desired, the 1-2-3 GNSO SOLVE code could be buffered into core. This would save a factor of 2–3 in storage, but cause an unknown degradation in execution speed. An alternative approach would be to not generate the SOLVE code but to use a scheme similar to that suggested by Chang [9]. This would save a factor of 3 in storage, and perhaps a factor of 2 in speed would be lost. (However, with this method, variability type cannot be exploited.) Thus a 7500 branch network transient analysis would be possible and a rough lower bound on execution time would be about 4 h on the 360/91.

ACKNOWLEDGMENT

The authors would like to thank T. E. Grapes, who was responsible for programming the compiler which converts GNSO to 1-2-3 GNSO, and D. A. Calahan, for extensive discussions and the interchange of ideas.

REFERENCES

[1] F. G. Gustavson, W. Liniger, and R. A. Willoughby, "Symbolic generation of an optimal Crout algorithm for sparse systems of linear equations," *J. Ass. Comput. Mach.*, vol. 17, pp. 87–109, January 1970.

[2] R. K. Brayton, F. G. Gustavson, and R. A. Willoughby, "Some results on sparse matrices," to be published in *Math. Comput.*

[3] W. F. Tinney and J. W. Walker, "Direct solutions of sparse network equations by optimally ordered triangular factorization," *Proc. IEEE*, vol. 55, pp. 1801–1809, November 1967.

[4] *Proc. Sparse Matrix Symp.*, IBM Rep. RA-1, March 1969.

[5] G. D. Hachtel and R. A. Rohrer, "Techniques for the optimal design and synthesis of switching circuits," *Proc. IEEE*, vol. 55, pp. 1864–1877, November 1967.

[6] F. H. Branin, Jr., "Computer methods of network analysis," *Proc. IEEE*, vol. 55, pp. 1787–1801, November 1967.

[7] C. W. Gear, "The automatic integration of stiff O.D.E.'s," *Proc. 1968 IFIPS Congr.*, pp. A81–A85.

[8] ——, "The control of parameters in the automatic integration of ordinary differential equations," *J. Ass. Comput. Mach.*; Department of Computer Science, University of Illinois, Urbana, Internal Rep., p. 14, May 1968.

[9] S. Sedore *et al.*, "SCEPTRE—an automated digital computer program for determining the response of electronic systems to transient nuclear radiation," vol. 11, IBM Space Guidance Center, Oswego, N. Y., Tech. Rep. AFWL-TR-66-126.

[10] E. S. Kuh and R. A. Rohrer, "The state-variable approach to network analysis, *Proc. IEEE*, vol. 53, pp. 672–686, July 1965.

[11] R. Fletcher and M. J. D. Powell, "A rapidly convergent descent method for minimization," *Comput. J.*, vol. 6, pp. 163–168, 1965.

[12] A. Nordsieck, "On numerical integration of ordinary differential equations," *Math. Comput.* vol. 16, pp. 22–49, 1962.

[13] D. A. Calahan, "Optimization of switching circuits," presented at the 2nd Biennial Cornell Conf. Engineering Applications of Electronic Phenomena, 1969.

[14] H. M. Markowitz, "The elimination form of the inverse and its application to linear programming," *Management Sci.*, vol. 3, pp. 255–269, April 1957.

[15] G. B. Dantzig, R. P. Harvey, R. D. McKnight, and S. S. Smith, "Sparse matrix techniques in two mathematical programming codes," *Proc. Sparse Matrix Symp.*, IBM Rep. RA-1, pp. 85–99, March 1969.

[16] P. D. Crout, "A short method for evaluating determinants and solving systems of linear equations with real or complex coefficients," *AIEE Trans.* (Supplement), vol. 60, pp. 1235–1240, December 1941.

[17] J. H. Wilkinson, *The Algebraic Eigenvalue Problem.* New York: Oxford University Press, 1965.

[18] ——, *Rounding Errors in Algebraic Processes.* Englewood Cliffs, N. J.: Prentice-Hall, 1963, pp. 121–126.

[19] S. Seshu, "Topological considerations in the design of driving point functions," *Proc.* AMS, pp. 1068–1073, October 1965.

[20] A. Chang, "Applications of sparse matrix methods in electric power system analysis," *Proc. Sparse Matrix Symp.*, IBM Rep. RA-1, pp. 113–121, March 1969.